NPM stands for Node Package Manager and it is a package manager for the JavaScript programming language. It is used primarily for installing and managing dependencies required by Node.js applications

Express is a popular web application framework for Node.js.One of the key benefits of Express is its flexibility. It allows developers to easily customize and extend the framework to meet the needs of their application.

Mongoose->Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It provides a straightforward schema-based solution to model data, validate, and interact with MongoDB.Mongoose allows developers to define schemas for their MongoDB collections and provides an API to perform CRUD (create, read, update, and delete) operations on those collections.

*mongoose*.connect("mongodb://0.0.0.0:27017/chat",{useNewUrlParser:true}).then(()=>{

    console.log("DB connection succesfully");

}).catch((*err*)=>{

    console.log(*err*.message);

});

Nodemon

Socket.io

Bcrypt

Cors

Dotenv

The line **const app=express()** is a common code snippet in Node.js applications that use the Express.js framework. **const app=express()** is the first step in creating a Node.js web application using the Express.js framework, and it creates an instance of the Express application that can be used to define routes and middleware.

The **body-parser** is a middleware for parsing the request body in the Express.js framework. It extracts the entire body portion of an incoming request and exposes it in a format that is easier to work with. The **body-parser** module is responsible for parsing the JSON, raw, text, and URL-encoded data submitted using HTTP POST request.

**public chat app src pages register**

event.preventDefault()->In web development, event.preventDefault() is a method used to prevent the default behavior of a browser event from occurring. When it comes to form submissions, this method is typically used to prevent the browser from automatically submitting the form and reloading the page.

**public chat app src pages app**

React is a popular JavaScript library for building user interfaces. One of the key features of React is its ability to manage state. State refers to the data that describes the current state of the user interface. React provides a way to manage state in a way that is easy to reason about and maintain.

<BrowserRouter> is a component provided by the React Router library, which is a popular routing solution for React applications.

<BrowserRouter> is used to wrap the entire application and provide routing functionality to it. It is used to define the root of the application's routing hierarchy and handle the dynamic rendering of components based on the URL of the page.

Routing is the process of determining how an application responds to a specific request for a particular endpoint or URL (Uniform Resource Locator). In web development,routing is used to define the navigation paths and URLs for the different pages or views within a web application.

This code is an example of a styled component in React using the styled-components library.

const FormContainer = styled.div` ->FormContainer is a custom component defined as a div element that is styled using the styled-components library. The backticks ( ) surrounding the div indicate that the styling is defined using a template literal, which allows for interpolation of JavaScript expressions within the CSS.

**useState** is a hook that enables functional components to have state, which was previously only possible with class components. When we call **useState**, it returns an array with two values: the current state value and a function to update that value. We can use destructuring to assign these values to variables and use them in our component.

useState is a built-in hook in React that allows you to add state to functional components. State is a way to store and manage data in a component that can change over

**useEffect** is a hook that allows us to perform side effects in functional components. Side effects can include things like fetching data from an API, updating the DOM, or subscribing to events. We can think of **useEffect** as a way to run code after the component has rendered.

useEffect is a built-in hook in React that allows you to perform side effects in functional components. Side effects are any actions that affect something outside the

component, such as fetching data from an API, updating the DOM, or subscribing to an event.

time, and useState provides a simple and convenient way to manage this state.

We use usestate to store the the values of username email password and confirmPassword and then set the value using the handle change function we destructure the values

Handle Validation using toast

Toastify is a popular library in React that allows developers to easily display notification messages in their applications. It can also be used to display validation messages when handling user input.

import { toast } from 'react-toastify';

import 'react-toastify/dist/ReactToastify.css';

Then, we can use the **toast.error()** method to display an error message if the validation fails

**app.use("/api/auth",userRoutes)**-> In a Node.js/Express.js application, the statement `app.use("/api/auth", userRoutes)` is used to register a set of routes to handle requests that start with a specific base URL.

Here, `app` is the instance of the Express application, and `use()` is a method that is used to add middleware to the application's request handling chain.

The first argument to `use()` is the base URL path, which in this case is "/api/auth". This means that any incoming requests that start with "/api/auth" will be passed to the `userRoutes` router for handling.

`userRoutes` is a set of routes that are defined in a separate file using the `express.Router()` method. These routes define the behavior for handling different HTTP methods (e.g. GET, POST, etc.) on specific URL paths.

By registering `userRoutes` with the `app` instance using `app.use()`, the application is able to handle requests that start with "/api/auth" and route them to the appropriate handler functions defined in `userRoutes`.

Now we will store the the entries /data in a mongoose schema by adding feature like min length of username should be 3 and so on. We will set the avatar image

In user controller .jsx we check if username or email already present or not, we then bcrypt the password it’s like hashing the password

${host}/api/auth/

The api/auth endpoint is used to authenticate users and obtain an access token. This endpoint is typically used by client applications to access protected resources on the server.The "api/auth" endpoint is typically used in web applications to authenticate users and provide access to secured resources

The api/auth endpoint typically supports the following HTTP methods:

* POST: This method is used to authenticate a user and obtain an access token.
* GET: This method is used to refresh an access token.
* DELETE: This method is used to revoke an access token.

This is a string that likely represents the base URL for an API endpoint. Here's a breakdown of the code:

${host} is a variable or constant that contains the hostname or IP address of the server that hosts the API.

/api/auth/ is a string that represents the endpoint path for the authentication-related APIs. This could include APIs for user registration, login, logout, password

reset, and other related functionalities.

When the two parts are concatenated using the ${} syntax, the resulting string is the full URL for the authentication-related API endpoint.

For example, if host is set to https://example.com, then the resulting URL for the authentication-related API endpoint would be https://example.com/api/auth/.

JavaScript

JavaScript Promises are a way to handle asynchronous operations and provide a more readable and predictable way to work with asynchronous code.

A Promise is an object that represents a value that may not be available yet, but will be resolved at some point in the future.

Use the JavaScript function JSON.parse() to convert text into a JavaScript object:

Convert a JavaScript object into a string with JSON.stringify().

Axios is a popular JavaScript library used to make HTTP requests from a web browser or Node.js. It allows developers to make asynchronous HTTP requests to REST endpoints and perform CRUD (create, read, update, delete) operations. Axios supports various features such as interceptors, cancellation, progress tracking, and automatic conversion of response data to JSON. It also provides easy-to-use methods for handling common HTTP methods such as GET, POST, PUT, and DELETE.

Axios used to take current user from all other user. // Example

server index.js

line 14->app.use("/api/auth", userRoutes) is a code snippet in a Node.js application that sets up a middleware to handle HTTP requests for user authentication.

In this code snippet, app refers to an instance of the Express.js framework, which is commonly used to build web applications in Node.js. The use() method is a built-in

function in Express.js that is used to register middleware functions.

The first parameter of app.use() is a string that defines the route path for the middleware. In this case, it is set to "/api/auth", which means that the middleware

will be triggered for any HTTP requests that match this path, such as https://example.com/api/auth/login or https://example.com/api/auth/register.

The second parameter, userRoutes, is a reference to a module that contains the middleware functions to handle the authentication requests. This module is typically

defined in a separate file and exported as a Node.js module using the module.exports statement.

AVATAR

const { data } = await axios.post(`${setAvatarRoute}/${user.\_id}`, {

image: avatars[selectedAvatar],

});

->

This is a code snippet using the Axios library to make an HTTP POST request to a server endpoint, sending an image to set as the user's avatar. Here's a breakdown of

the code:

axios.post is a method provided by the Axios library that sends an HTTP POST request to a specified endpoint. The first argument to the method is the URL of the

endpoint to send the request to, which is constructed using the setAvatarRoute variable and the user's ID (user.\_id).

The second argument to the axios.post method is an object that contains the data to be sent in the body of the request. In this case, it contains an image property,

whose value is an image object selected from the avatars array at the selectedAvatar index.

The await keyword is used before the axios.post method call to wait for the response from the server before continuing. The response data is then destructured using

the object destructuring syntax and assigned to a data variable.

req.params.id is a property in the Node.js req object that is commonly used in web application frameworks like Express.

It refers to a route parameter in the URL path of an incoming HTTP request. For example, if you have a route defined as /users/:id, and the client makes a GET request

to /users/123, req.params.id would be equal to "123".

You can use req.params.id to retrieve the value of the id parameter and use it in your server-side code to perform actions such as fetching data from a database or

returning specific content to the client.

message model

This block defines the fields of the schema.

message is an object containing a single field text, which is a required string value.

users is a field of type Array, which may contain any type of data.

sender is a field of type mongoose.Schema.Types.ObjectId, which represents a MongoDB ObjectId that references a User object. This field is required.

Sure, I'd be happy to explain this code line by line. This code defines a Mongoose schema for a message in a chat application.

arduino

Copy code

const MessageSchema = mongoose.Schema(

This line declares a new Mongoose schema called MessageSchema using the mongoose.Schema() method.

Copy code

{

message: {

text: { type: String, required: true },

},

users: Array,

sender: {

type: mongoose.Schema.Types.ObjectId,

ref: "User",

required: true,

},

},

This block defines the fields of the schema.

message is an object containing a single field text, which is a required string value.

users is a field of type Array, which may contain any type of data.

sender is a field of type mongoose.Schema.Types.ObjectId, which represents a MongoDB ObjectId that references a User object. This field is required.

This block defines the options for the schema. In this case, the timestamps option is set to true, which tells Mongoose to automatically create two fields in the

schema called createdAt and updatedAt that store timestamps of when the document was created and last updated, respectively.

Chat container ->header chat container chat input

Chat header->current chat name with avatar

Chat container->

Chat input->emoji picker react , import IoMdsend , Picker,BsEmojiSmileFill and submit button after input .

Messages are store in messageControllers in which we are adding messages in a variable from, to, message

In getMessages variable are from, to and sort all the messages with updatedAt

**Socket.io**

Socket.I0 enables real-time bidirectional event-based communication. It works on every platform, browser or device, focusing equally on reliability and speed. Socket.I0 is built on top of the WebSockets API (Client side) and Node js.

Socket.I0 is a JavaScript library for real-time web applications.It enables real-time, bi-directional communication between web clients and servers. It has two parts - a client-side library that runs in the browser, and a server-side library for node.js. Both components have an identical API.

A real-time application (RTA) is an application that functions within a period that the user senses as immediate or current.

Some examples of real-time applications are -

• Instant messengers - Chat apps like Whatsapp, Facebook Messenger, etc. You need not refresh your app/website to receive new messages.

• Push Notifications - When someone messages you on instagram, you receive a notification

• Online Gaming - Games like Valorant ,Counter Strike,etc.

**Why Socket.io**

Socket.IO is quite popular, it is used by Microsoft Office, Yammer, Zendesk, Trello.. numerous other organizations to build robust real-time systems. It one of the most powerful JavaScript frameworks on GitHub, and most depended-upon NPM (Node Package Manager) module. Socket. IO also has a huge community, which means help is quite easy

* **socket.send()** method sends a message over the socket connection. It takes one argument, which is the message to be sent, and sends it to the server.
* **socket.emit()** method allows you to emit custom events from the client to the server. It takes two arguments - the name of the event to be emitted and any data that should be sent along with the event.
* **socket.on()** method is used to listen for events on the socket connection. It takes two arguments - the name of the event to listen for and a callback function to be called when the event is triggered.
* **socket.in()** method is used to send a message to a specific room. It takes one argument, which is the name of the room, and sends the message to all sockets that have joined that room.

**io.on()** is used on the server-side to listen for events from clients. When a client emits an event, the server can use **io.on()** to listen for that event and perform some action.

**socket.on()** is used on the client-side to listen for events from the server. When the server emits an event to a specific client, the client can use **socket.on()** to listen for that event and perform some action.

In React, useRef is a hook that creates a reference to a mutable value that persists throughout the component's life cycle. It is similar to creating a reference to an HTML element using document.getElementById() in traditional web development.

In Socket.IO, **socket.to(room)** is used to send a message or perform an action to all the connected clients in a specific room.

When a socket joins a room using **socket.join(room)**, the server keeps track of all sockets that are in that room. Using **socket.to(room)** we can send an event or message to all the sockets in that room.

Here, **socket.to(room)** selects all the sockets in the given room, and **emit('message', data)** sends the message with the given data to all the selected sockets.

* to : The to method is used to specify the destination of a message. The destination can be a socket, a port, or an IP address.
* in : The in method is used to receive a message. The message will be stored in a buffer.
* out : The out method is used to send a message. The message will be sent to the destination specified by the to method.
* emit : The emit method is used to send a message to all connected sockets. The message will be received by all sockets that have subscribed to the event.

Explain ->useEffect(() => { arrivalMessage && setMessages((prev) => [...prev, arrivalMessage]); }, [arrivalMessage]);

This is a React `useEffect` hook that is used to update the state of `messages` in response to the arrival of a new message.

The hook takes a function as its first parameter, which is executed whenever any of the dependencies specified in the second parameter changes. In this case, the dependency is `arrivalMessage`.

The function inside the `useEffect` hook is checking if `arrivalMessage` is truthy (i.e., not null or undefined). If `arrivalMessage` is not null, it appends the new message to the existing messages array by using the spread operator (`...`) and the `prev` parameter of the `setMessages` function to get the previous value of the messages state.

The `[arrivalMessage]` part of the hook specifies that the effect should only be triggered when `arrivalMessage` changes, ensuring that the component is not unnecessarily re-rendered if some other state value is updated.

Explain ->`scrollRef.current?.scrollIntoView({ behavior: "smooth" });`

is used in React to scroll to a specific element in the UI smoothly.

Here, `scrollRef` is a reference to the DOM element that needs to be scrolled into view. `scrollIntoView` is a method that is available on all DOM elements and is used to scroll an element into view.

The `behavior` option is set to "smooth" to enable smooth scrolling animation. If `behavior` is not set, the scrolling will be instant.

Using the optional chaining operator (`?.`) ensures that the `scrollIntoView` method is only called if `scrollRef.current` is not null or undefined. This prevents a runtime error in case `scrollRef.current` is null or undefined.